Multilayer NN + Gradient Descent

AW

Lecture Overview

1. **Multilayer Networks**

2. **Gradient-based optimization**

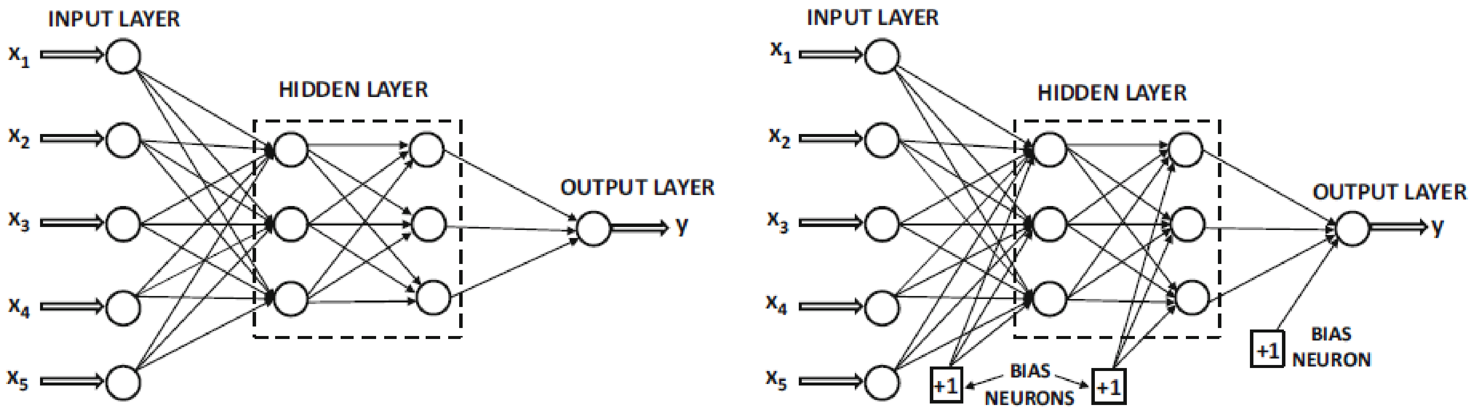
Feed-forward Networks: Terminology

- Every element which holds an input and has output is called a 'neuron'. A shallow neural network consists of two or three layers, anything more than that is considered deep.

  Example of shallow network: Perceptron contains an input and output layer, of which the output layer is the only computation performing layer. It is a shallow neural network

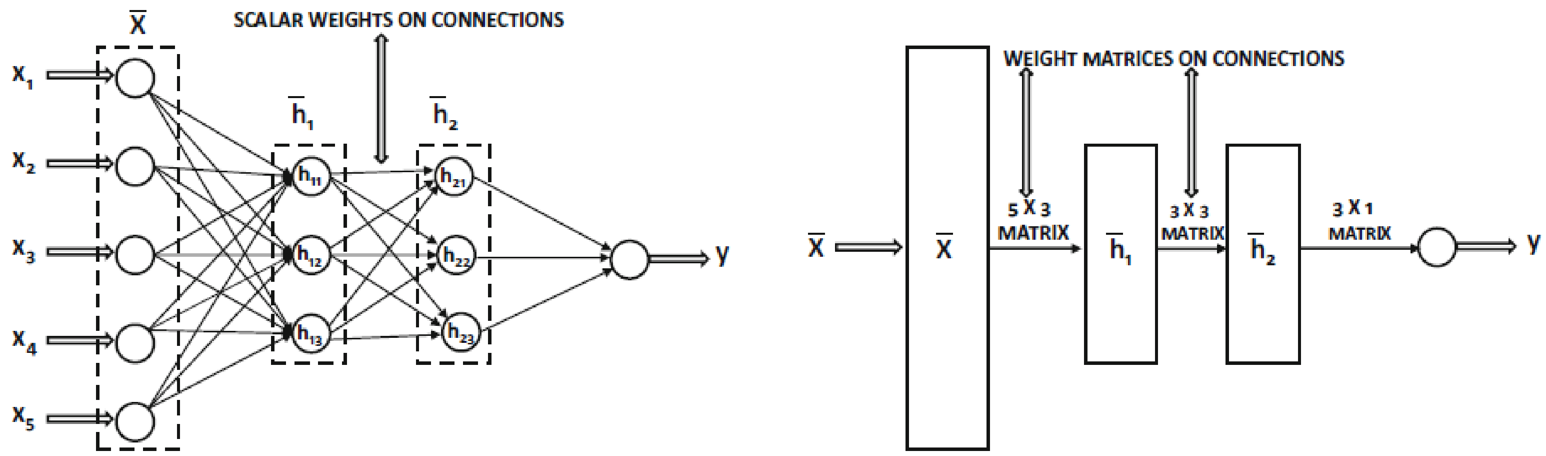
- The architecture of multilayer neural networks is referred to as *feed-forward* networks, when successive layers feed into one another in the forward direction from input to output. The default architecture of feed-forward networks assumes that all nodes in one layer are connected to those of the next layer.
- Feed-forward networks are also known as *Multi-Layer Perceptron*

Examples of Multilayer Networks



- Neural networks may use neurons with or without constant bias. Bias neurons can be used both in the hidden layers and in the output layers.
- The neural network is almost fully defined by
  1. The number of layers
  2. The number and type (weight vector and activation function) of nodes in each layer
  3. The loss function that is optimized in the output layer.

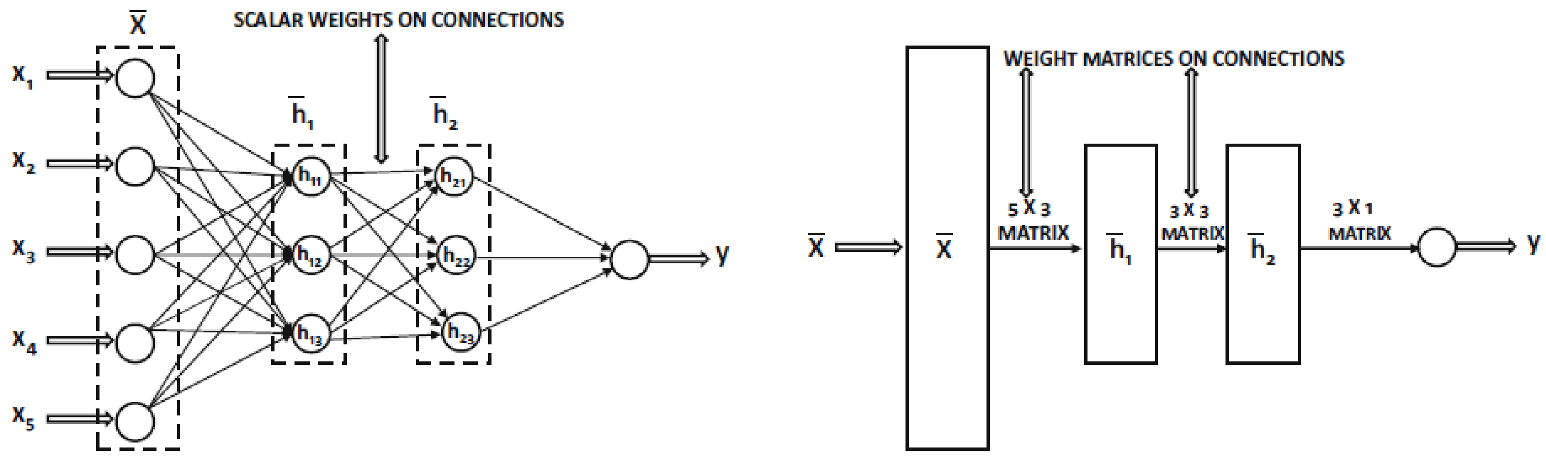
Network Graphs and Their Matrix Representation



- If a neural network contains $p_1, \ldots, p_k$ units in each of its k layers, then the (column) vector representations of these outputs (denoted by $\vec{h}_1, \ldots, \vec{h}_k$) have dimensionalities $\dim \vec{h}_1 = p_1, \ldots, \dim \vec{h}_k = p_k$.

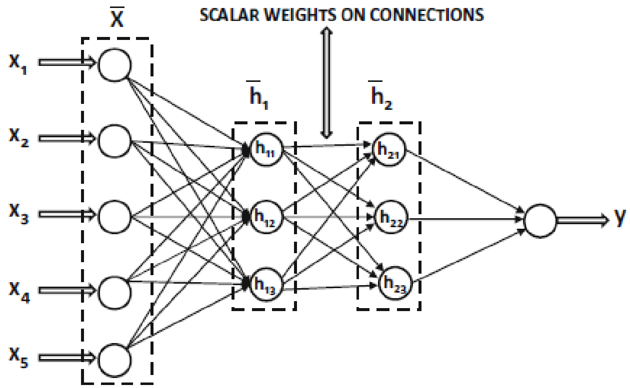
  Example: for the net on the figure $\dim \vec{h}_1 = \dim \vec{h}_2 = 3$
- The number of units in each layer is referred to as the *dimensionality* of that layer

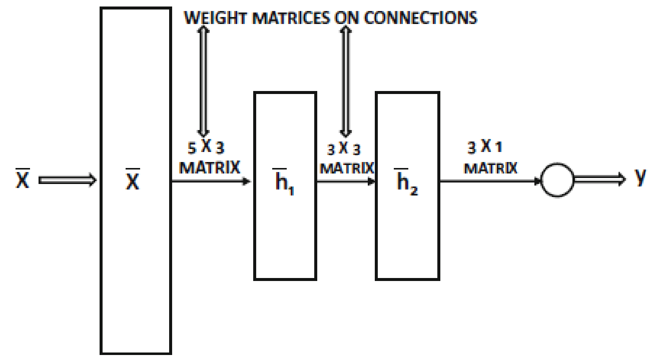
NN Matrix Representation (continued)



- The weights of the connections between the input layer and the first hidden layer are contained in a *matrix* W_1 with size $d \times p_1$, where d is the number of inputs into network, column $\vec{w}_i^1$ contains the weights of inputs into Σ-part of i^{th} neuron in layer 1
- The weights between the r^{th} hidden layer and the $(r + 1)^{st}$ hidden layer are given in the $p_r \times p_{r+1}$ matrix W_r in which column $\vec{w}_i^r$ defines the weights of inputs into Σ-part of i^{th} neuron in layer r

NN Matrix Representation - Example





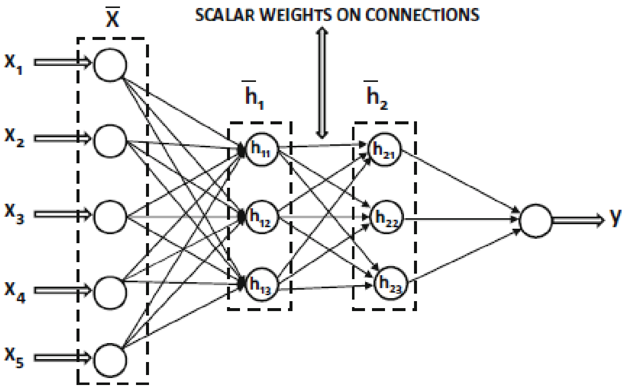
Let weight vectors for neurons h_{11}, h_{12}, h_{13} be given by their Σ-parts

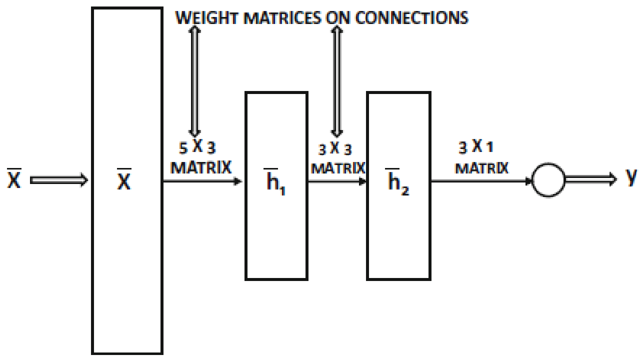
$$\begin{array}{c} 1x_1 + 2.5x_2 + 2x_3 + 2.5x_4 + 1x_5 \\ 2x_1 + 5x_2 + 2x_3 + 1x_4 + 2x_5 \quad , \\ 1x_1 + 3x_2 + 4x_3 + 3x_4 + 1x_5 \end{array}$$

Then

$$W_1 = \begin{pmatrix} 1 & 2 & 1 \\ 2.5 & 5 & 3 \\ 2 & 2 & 4 \\ 2.5 & 1 & 3 \\ 1 & 2 & 1 \end{pmatrix}$$

NN Matrix Representation - Example





Let weight vectors for neurons h_{11}, h_{12}, h_{13} be given by their Σ-parts

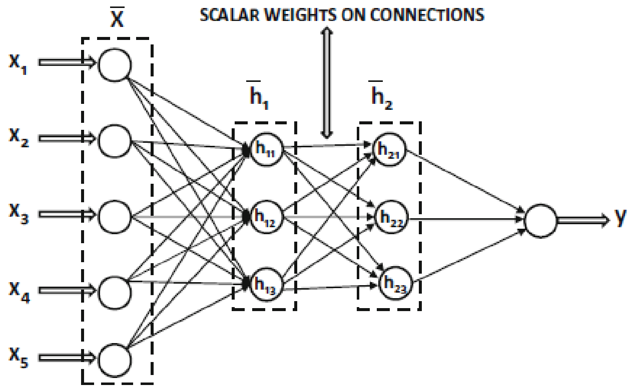
$$\begin{array}{c} 1x_1 + 2.5x_2 + 2x_3 + 2.5x_4 + 1x_5 \\ 2x_1 + 5x_2 + 2x_3 + 1x_4 + 2x_5 \quad , \\ 1x_1 + 3x_2 + 4x_3 + 3x_4 + 1x_5 \end{array}$$

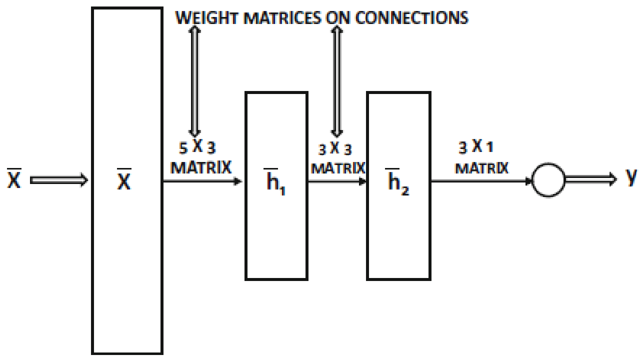
and weight vectors for input of neurons h_{21}, h_{22}, h_{23} be given by their Σ-parts

$$\begin{array}{c} 1h_{11} + 2h_{12} + 3h_{13} \\ 2h_{11} + 3h_{12} + 1h_{13} \, , \\ 1h_{11} + 2h_{12} + 1h_{13} \end{array}$$

. Then

$$W_1 = \begin{pmatrix} 1 & 2 & 1 \\ 2.5 & 5 & 3 \\ 2 & 2 & 4 \\ 2.5 & 1 & 3 \\ 1 & 2 & 1 \end{pmatrix}, W_2 = ?$$





Let weight vectors for neurons h_{11}, h_{12}, h_{13} be given by their Σ-parts

$$\begin{gathered} 1x_1 + 2.5x_2 + 2x_3 + 2.5x_4 + 1x_5 \\ 2x_1 + 5x_2 + 2x_3 + 1x_4 + 2x_5 \quad , \\ 1x_1 + 3x_2 + 4x_3 + 3x_4 + 1x_5 \end{gathered}$$

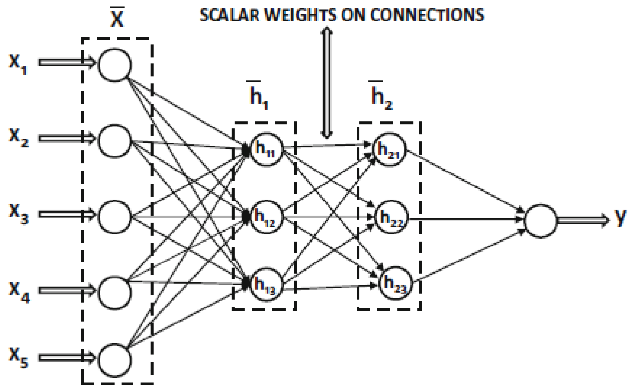
and weight vectors for neurons h_{21}, h_{22}, h_{23} be given by their Σ-parts

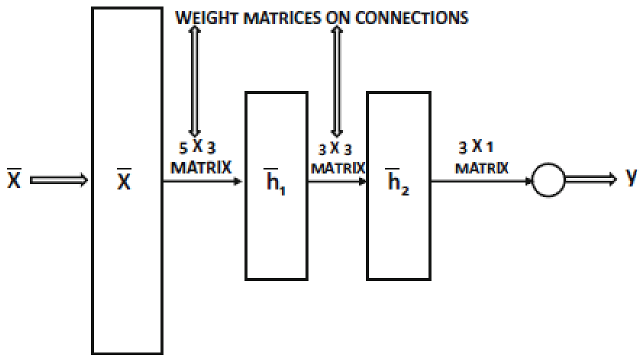
$$\begin{gathered} 1h_{11} + 2h_{12} + 3h_{13} \\ 2h_{11} + 3h_{12} + 1h_{13} \, , \\ 1h_{11} + 2h_{12} + 1h_{13} \end{gathered}$$

and weights of output neuron be given by $4h_{21} + 2h_{22} + 3h_{23}$. Then

$$W_1 = \begin{pmatrix} 1 & 2 & 1 \\ 2.5 & 5 & 3 \\ 2 & 2 & 4 \\ 2.5 & 1 & 3 \\ 1 & 2 & 1 \end{pmatrix}, W_2 = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 3 & 2 \\ 3 & 1 & 1 \end{pmatrix} \text{ and } W_o = ?$$

NN Matrix Representation - Example





Let weight vectors for neurons h_{11}, h_{12}, h_{13} be given by their Σ-parts

$$\begin{gathered} 1x_1 + 2.5x_2 + 2x_3 + 2.5x_4 + 1x_5 \\ 2x_1 + 5x_2 + 2x_3 + 1x_4 + 2x_5 \quad , \\ 1x_1 + 3x_2 + 4x_3 + 3x_4 + 1x_5 \end{gathered}$$

and weight vectors for neurons h_{21}, h_{22}, h_{23} be given by their Σ-parts

$$\begin{gathered} 1h_{11} + 2h_{12} + 3h_{13} \\ 2h_{11} + 3h_{12} + 1h_{13} \, , \\ 1h_{11} + 2h_{12} + 1h_{13} \end{gathered}$$

and weights of output neuron be given by $4h_{21} + 2h_{22} + 3h_{23}$. Then

$$W_1 = \begin{pmatrix} 1 & 2 & 1 \\ 2.5 & 5 & 3 \\ 2 & 2 & 4 \\ 2.5 & 1 & 3 \\ 1 & 2 & 1 \end{pmatrix}, W_2 = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 3 & 2 \\ 3 & 1 & 1 \end{pmatrix} \text{ and } W_o = \begin{pmatrix} 4 \\ 2 \\ 3 \end{pmatrix}$$

NN Matrix Representation - continued

The neural network is a transformation that takes d-dimensional input vector $\vec{x}$ and transforms it into the output using the following recursive equations:

$h_1 = \Phi(W_1^T \vec{x})$ [Input to Hidden Layer]

$h_{p+1} = \Phi\left(W_{p+1}^T \vec{h}_p\right) \forall p \in \{1, \ldots, k - 1\}$ [Hidden to Hidden Layer]

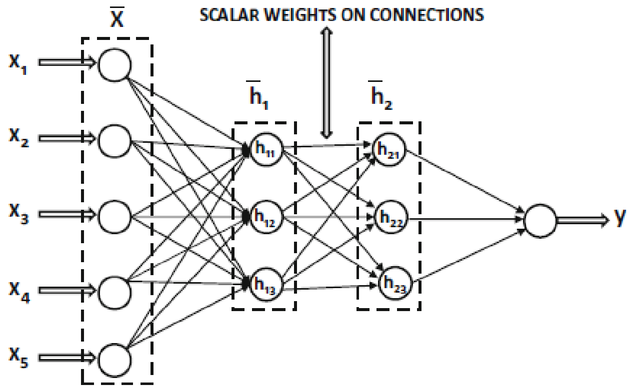
$o = \Phi(W_{k+1}^T \vec{h}_k)$ [Hidden to Output Layer]

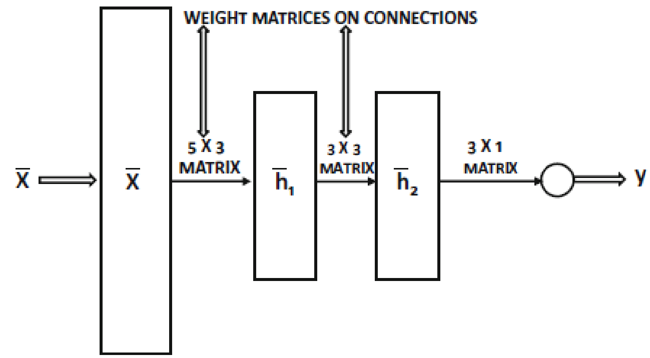
where the activation functions (like the ReLU or sigmoid) are applied in *element-wise* fashion to their vector arguments, i.e. $\Phi\left(\begin{bmatrix} v_1 \\ \vdots \\ v_k \end{bmatrix}\right) = \begin{bmatrix} \Phi(v_1) \\ \vdots \\ \Phi(v_k) \end{bmatrix}$

Notes:

1. It is implicitly assume that all neurons within a layer have the same activation function (though it is not a requirement)
2. some activation functions such as the softmax (which are typically used in the output layers) naturally have vector arguments.

NN Matrix Representation - Example





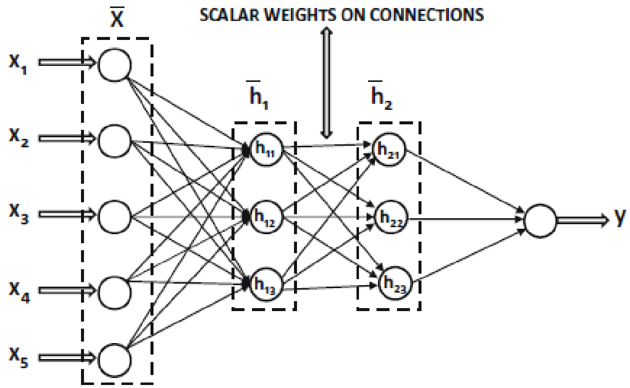
$$W_1 = \begin{pmatrix} 1 & 2 & 1 \\ 2.5 & 5 & 3 \\ 2 & 2 & 4 \\ 2.5 & 1 & 3 \\ 1 & 2 & 1 \end{pmatrix}, W_2 = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 3 & 2 \\ 3 & 1 & 1 \end{pmatrix} \text{ and } W_o = \begin{pmatrix} 4 \\ 2 \\ 3 \end{pmatrix}$$

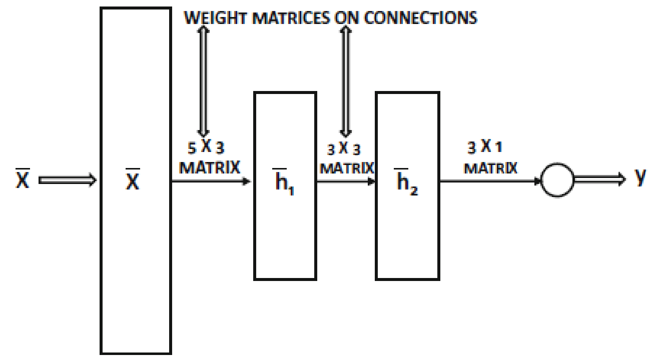
Let $\Phi_1(v) = \max\{v,0\}$ $(ReLU)$, $\Phi_2(v) = \text{sign(v)}$ (step), $\Phi_3(v) = v$ (linear)

Let input $\vec{x} = \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \\ 1 \end{pmatrix}$. Then $v_1 = W_1^T \vec{x} = \begin{pmatrix} 1 & 2 & 1 \\ 2.5 & 5 & 3 \\ 2 & 2 & 4 \\ 2.5 & 1 & 3 \\ 1 & 2 & 1 \end{pmatrix}^T \begin{pmatrix} 1 \\ -1 \\ 3 \\ -1 \\ -2 \end{pmatrix} = \begin{pmatrix} 0 \\ -3 \\ 5 \end{pmatrix}$,

and $\vec{\mathrm{h}}_1 = \Phi_1(v_1) = \begin{pmatrix} \max(0,0) \\ \max(0,-3) \\ \max(0,5) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 5 \end{pmatrix}$

NN Matrix Representation - Example



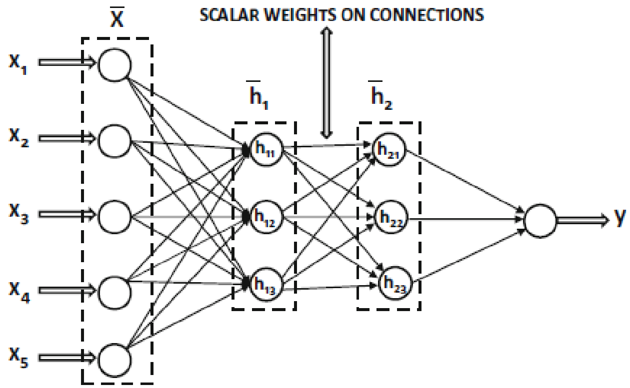


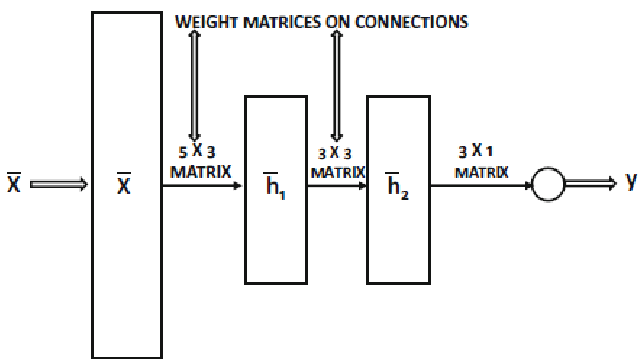
$$W_1 = \begin{pmatrix} 1 & 2 & 1 \\ 2.5 & 5 & 3 \\ 2 & 2 & 4 \\ 2.5 & 1 & 3 \\ 1 & 2 & 1 \end{pmatrix}, W_2 = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 3 & 2 \\ 3 & 1 & 1 \end{pmatrix} \text{ and } W_o = \begin{pmatrix} 4 \\ 2 \\ 3 \end{pmatrix}$$

Let $\Phi_1(v) = \max\{v{,}0\}$ $(ReLU)$, $\Phi_2(v) = \text{sign}(v)$ (step), $\Phi_3(v) = v$ (linear)

Let input $\vec{x} = \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \\ 1 \end{pmatrix}$. Then $v_1 = \begin{pmatrix} 0 \\ -3 \\ 5 \end{pmatrix}$, $\vec{h}_1 = \begin{pmatrix} 0 \\ 0 \\ 5 \end{pmatrix}$ and $v_2 = ?\,, h_2 = ?$

NN Matrix Representation - Example



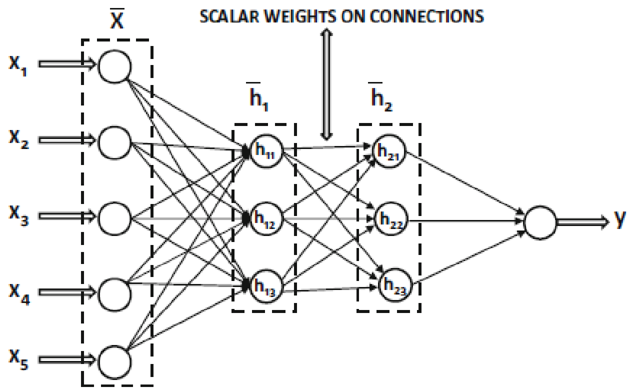


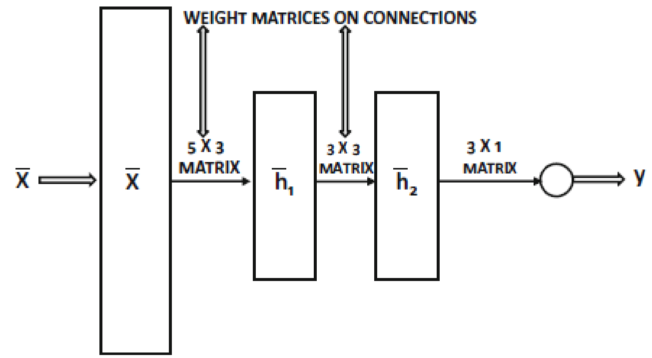
$$W_1 = \begin{pmatrix} 1 & 2 & 1 \\ 2.5 & 5 & 3 \\ 2 & 2 & 4 \\ 2.5 & 1 & 3 \\ 1 & 2 & 1 \end{pmatrix}, W_2 = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 3 & 2 \\ 3 & 1 & 1 \end{pmatrix} \text{ and } W_o = \begin{pmatrix} 4 \\ 2 \\ 3 \end{pmatrix}$$

Let $\Phi_1(v) = \max\{v, 0\}$ $(ReLU)$, $\Phi_2(v) = \text{sign(v)}$ (step), $\Phi_3(v) = v$ (linear)

Let input $\vec{x} = \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \\ 1 \end{pmatrix}$. Then $v_1 = \begin{pmatrix} 0 \\ -3 \\ 5 \end{pmatrix}$, $\vec{h}_1 = \begin{pmatrix} 0 \\ 0 \\ 5 \end{pmatrix}$ and $v_2 = \begin{pmatrix} 15 \\ 5 \\ 5 \end{pmatrix}$, $h_2 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$, so $y = ?$

NN Matrix Representation - Example





$$W_1 = \begin{pmatrix} 1 & 2 & 1 \\ 2.5 & 5 & 3 \\ 2 & 2 & 4 \\ 2.5 & 1 & 3 \\ 1 & 2 & 1 \end{pmatrix}, W_2 = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 3 & 2 \\ 3 & 1 & 1 \end{pmatrix} \text{ and } W_o = \begin{pmatrix} 4 \\ 2 \\ 3 \end{pmatrix}$$

Let $\Phi_1(v) = \max\{v,0\}$ $(ReLU)$, $\Phi_2(v) = \text{sign(v)}$ (step), $\Phi_3(v) = v$ (linear)

Let input $\vec{x} = \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \\ 1 \end{pmatrix}$. Then $v_1 = \begin{pmatrix} 0 \\ -3 \\ 5 \end{pmatrix}, \vec{h}_1 = \begin{pmatrix} 0 \\ 0 \\ 5 \end{pmatrix}$ and $v_2 = \begin{pmatrix} 15 \\ 5 \\ 5 \end{pmatrix}, h_2 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$, so $y = 9$

Typical Activations in Multilayer Networks

- Activation in output layer depends on the type of output. If
  - The intended output is a real-valued number then it is typically identity,
  - The intended output is in $\{0,1\}$ then it is typically sigmoid – i.e. output of the ANN is not 0/1 but the probability of 1
  - The intended output is belongs to a finite set then it is typically softmax – i.e. output of theANN is not an element of the set but a probability distribution on the output set!
    - Softmax is almost exclusively is used for output. It is always paired with *cross-entropy* loss.
- Hidden layer activations are almost always nonlinear
- Hidden neurons always use the same activation function over the entire layer of the network and often the same over the whole ANN.
  - Tanh often (but not always) preferable to sigmoid.
  - ReLU has largely replaced tanh and sigmoid in many applications.

Hidden Layers Must be Nonlinear!

- Suppose hidden layers are not nonlinear – so activation is identity
- *Claim*: multi-layer network that uses only the identity activation function in all layers reduces to a single-layer network that performs linear regression.
- $\vec{h}_1 = \Phi(W_1^T \vec{x}) = W_1^T \vec{x}$
- $\vec{h}_{p+1} = \Phi\left(W_{p+1}^T \vec{h}_p\right) = W_{p+1}^T \vec{h}_p \quad \forall p \in \{1, \ldots, k-1\}$
- $o = \Phi(W_{k+1}^T \vec{h}_k) = W_{k+1}^T \vec{h}_k$

Composition gives

$$o = W_{k+1}^T W_k^T \cdot \ldots \cdot W_1^T \vec{x}_1 = \underbrace{(W_1 W_2 \cdot \ldots \cdot W_{k+1})^T}_{W_{total}} \vec{x}$$

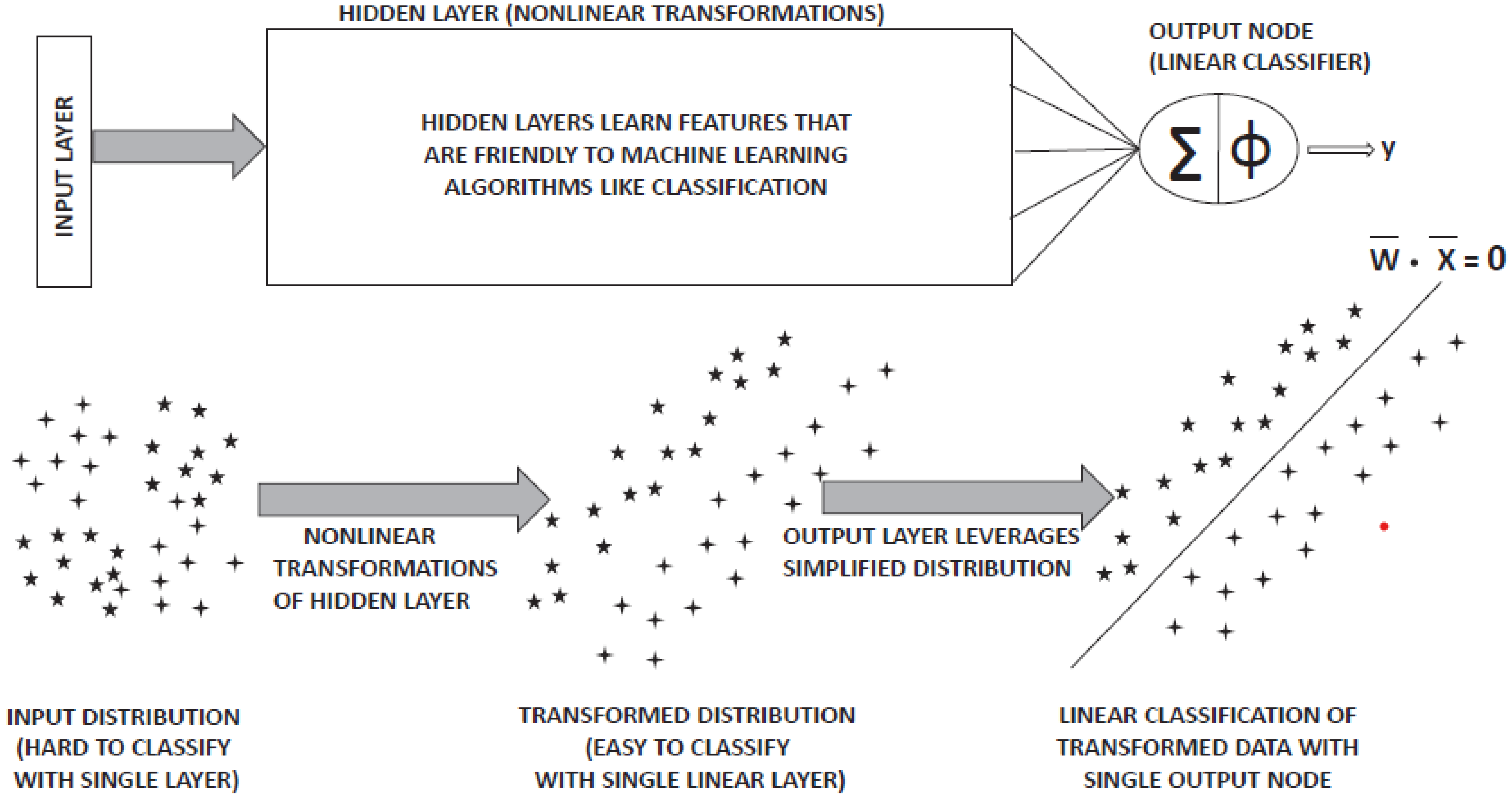
so it is equivalent to single layer network.

Role of Hidden Layers

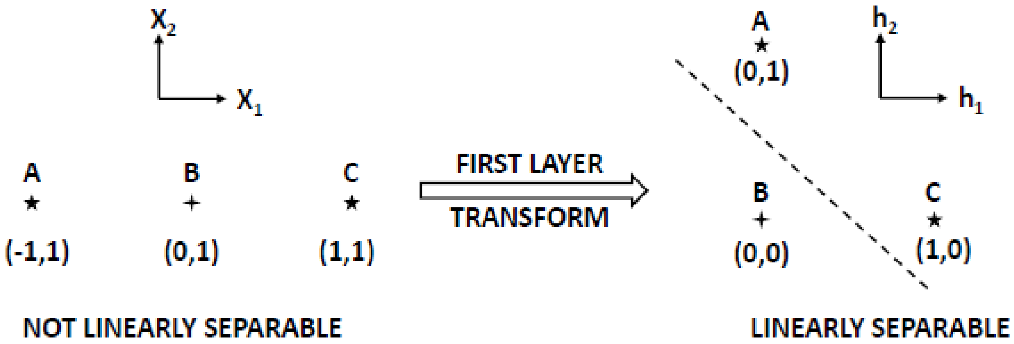
- Nonlinear hidden layers perform hierarchical feature selection/aggregation :
  - Early layers learn atomic features and later layers learn complex features

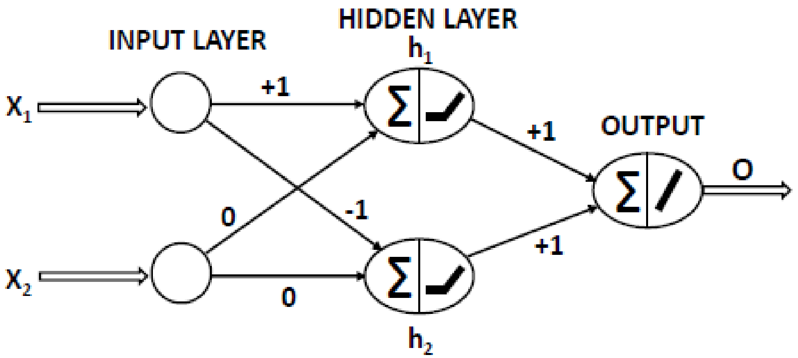
    Example. Image data:
    - Early layers learn elementary edges
    - Middle layers learn more contain complex features (e.g. honeycombs)
    - End layers contain complex features like a part of a face.
- The final output layer performs inference with transformed features

Schematic Depiction of Feature Engineering



Example: Linearly Inseparable Data





The hidden units have ReLU activation, and they learn the two new features h_1 and h_2 with linear separator

$$h_1 + h_2 = 0.5$$

where $h_1 = \max\{x_1, 0\}$ and $h_2 = \max\{-x_1, 0\}$

Lecture Overview

1. **Multilayer Networks**

2. **Gradient-based optimization**

Derivative and Gradient Descent - Intuition

- The derivative $f'(x) = \frac{df(x)}{dx}$ gives the slope of $f(x)$ at x, i.e. it specifies how to scale a small change in the input in order to obtain the corresponding change in the output:

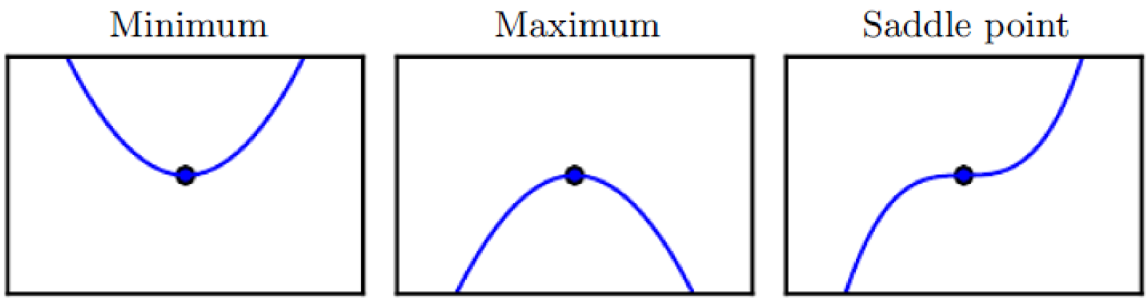
$$f(x+\varepsilon) \approx \ f(x) + \varepsilon \frac{df(x)}{dx}$$

- The derivative is useful for minimizing a function because it tells us how to change x to make a small improvement in y:

  $f\left(x - \varepsilon \cdot \operatorname{sign}\left(\frac{df(x)}{dx}\right)\right)$ is less than $f\ (x)$ for small enough ε, so when searching for minimum, we can reduce $f(x)$ by moving x in small steps with opposite sign of the derivative. This technique is called *gradient descent*.

- When $\frac{df(x)}{dx} = 0$ we have no information in which direction to move. The points where $\frac{df(x)}{dx} = 0$ are *stationary points:* *minimums, maximums,* and *saddle points*

Minimums, Maximums, and Saddle Points



- *Local minimum* is a point x where $f(x)$ is lower than at all neighboring points, so it is no longer possible to decrease $f(x)$ by making infinitesimal steps.
- *Local maximum* x is a point where $f\ (x)$ is higher than at all neighboring points, so it is not possible to increase $f\ (x)$ by making infinitesimal steps.
- Stationary points that are neither maxima nor minima and both increase and decrease by making infinitesimal steps are possible but which way which is not clear are *saddle points*.
- A point that obtains the absolute lowest (highest) value of $f(x)$ is a *global minimum* (resp. *global maximum*)

Partial, Directional Derivatives and Gradient

- Chain rule of taking derivatives: for $f(y(x))$ we have
$$[f(y(x)]'_x|_{x_0} = \frac{df}{dx}|_{x_0} = \frac{df}{dy}|_{y(x_0)} \cdot \frac{dy}{dx}|_{\mathrm{x}_0}$$
- We often minimize functions that have multiple inputs:

$f: \mathbb{R}^n \to \mathbb{R}$. For functions with multiple inputs, we need partial derivatives $\frac{\partial f(\vec{x})}{\partial x_i}|_{\vec{x}_0}$ to find minimums. It measures how f changes near point $\vec{x}_0$ when only coordinate x_i is increased by ε.
- To determine how f changes when 2 coordinates change we have second derivative: $\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i}\left(\frac{\partial f}{\partial x_j}\right) = \frac{\partial}{\partial x_j}\left(\frac{\partial f}{\partial x_i}\right)$
- Gradient of f is the vector $\nabla_{\vec{x}} f(\vec{x})|_{x_0}$ with entries - partial derivatives at point $\vec{x}_0$. Element i of the gradient is the partial derivative of f with respect to x_i.
- In multiple dimensions a point $\vec{x}_0$ is stationary if every element of the gradient is 0, i.e. $\nabla_{\vec{x}} f(\vec{x})|_{\vec{x}_0} = \vec{0}$.
- Chain rule applies to partial derivatives too. For $f(g_1(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n))$ we have :
$$\frac{\partial f}{\partial x_i} = \frac{\partial f}{\partial g_1} \cdot \frac{\partial g_1}{\partial x_i} + \frac{\partial f}{\partial g_2} \cdot \frac{\partial g_2}{\partial x_i} + \dots + \frac{\partial f}{\partial g_k} \cdot \frac{\partial g_k}{\partial x_i}$$

Example of Chain Rule for Partial Derviatives

Calculate $\frac{\partial f}{\partial u}$ given $f(x, y, z) = 3x^2 - 2xy + 4z^2$ where $x(u, v) = e^{u \cdot \sin v}$; $y(u, v) = e^{u \cdot \cos v}$; $z(u, v) = e^u$.

We need $\frac{\partial f}{\partial u} = \frac{\partial f}{\partial x} \cdot \frac{\partial x}{\partial u} + \frac{\partial f}{\partial y} \cdot \frac{\partial y}{\partial u} + \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial u}$

So we need to first compute

$\frac{\partial f}{\partial x} =?$ $\frac{\partial f}{\partial y} =?$ $\frac{\partial f}{\partial z} =?$

Recall that for $f(x) = ax^c$ where a and c are constants we have $f_x' = ac\, x^{c-1}$ or you can use Mathemtica/Wolfram Alpa

Example of Chain Rule for Partial Derviatives

Calculate $\frac{\partial f}{\partial u}$ given $f(x,y,z) = 3x^2 - 2xy + 4z^2$ where

$x(u,v) = e^{u \cdot \sin v}$; $y(u,v) = e^{u \cdot \cos v}$; $z(u,v) = e^{u}$.

We need $\frac{\partial f}{\partial u} = \frac{\partial f}{\partial x} \cdot \frac{\partial x}{\partial u} + \frac{\partial f}{\partial y} \cdot \frac{\partial y}{\partial u} + \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial u}$

We have:

$\frac{\partial f}{\partial x} = 6x - 2;$ $\frac{\partial f}{\partial y} = -2x;$ $\frac{\partial f}{\partial z} = 8z$

We need

$\frac{\partial x}{\partial u} = ?$ $\frac{\partial y}{\partial u} = ?$ $\frac{\partial z}{\partial u} = ?$

Recall that $f\big(g(x)\big)'_x = g(x)'_x f(g)'_g$ and $(e^x)'_x = e^x$

or use Mathematica/Wolfram alpha for all these equalities

Example of Chain Rule for Partial Derviatives

Calculate $\frac{\partial f}{\partial u}$ given $f(x, y, z) = 3x^2 - 2xy + 4z^2$ where

$x(u, v) = e^{u \cdot \sin v}$; $y(u, v) = e^{u \cdot \cos v}$; $z(u, v) = e^{u}$.

We have:

$$\frac{\partial f}{\partial x} = 6x - 2; \qquad \frac{\partial f}{\partial y} = -2x; \qquad \frac{\partial f}{\partial z} = 8z$$

$$\frac{\partial x}{\partial u} = \sin v \cdot e^{u \cdot \sin v}; \qquad \frac{\partial y}{\partial u} = \cos v \cdot e^{u \cdot \cos v}; \qquad \frac{\partial z}{\partial u} = e^{u};$$

Then

$$\frac{\partial f}{\partial u} = \frac{\partial f}{\partial x} \cdot \frac{\partial x}{\partial u} + \frac{\partial f}{\partial y} \cdot \frac{\partial y}{\partial u} + \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial u}$$

$$= \left(6e^{u \cdot \sin v} - 2\right) \sin v \cdot e^{u \sin v} - 2e^{u \cdot \sin v} \cos v \cdot e^{u \cdot \cos v} + 8e^{u} e^{u}$$

Directional Derivative and its Use

- directional derivative of $f(\vec{x})$ in direction $\vec{u}$ (where $\|u\| = 1$) is the slope of the function f in direction $\vec{u}$, i.e. it is a derivative of a function $f(\vec{x} + \alpha\vec{u})$ at a point $\vec{x}_0 + \alpha\vec{u}$ when $\alpha \to 0$ (i.e. taken with respect to α). Using chain rule

$$\frac{\partial}{\partial\alpha} f(\vec{x} + \alpha\vec{u})|_{\alpha=0} = \vec{u}^T \nabla_{\vec{x}} f(x) = \vec{u} \cdot \nabla_{\vec{x}} f(\vec{x})$$

- To minimize f, we'd like to use the direction in which f decreases the fastest. Using the directional derivative:
- $\min_{\vec{u},\|u\|=1} \vec{u} \cdot \nabla_{\vec{x}} f(x) = \min_{\vec{u},\|u\|=1} \|\vec{u}\| \| \nabla_{\vec{x}} f(x)\| \cos\theta$ where θ is an angle between u and $\nabla_{\vec{x}} f(x)$ (recall that $\frac{a \cdot b}{\|a\|\|b\|} = \cos\theta$). Since it is required $\|\vec{u}\| = 1$ and $\|\nabla_{\vec{x}} f(x)\|$ does not depend on $\vec{u}$ we get $\min_{\vec{u},\|u\|=1} \|\vec{u}\| \| \nabla_{\vec{x}} f(x)\| \cos\theta = \min_{\vec{u}} \cos\theta$ which is at $\min = -1$ when u points in the opposite direction from gradient!

Gradient Descent

- Decreasing f by moving in the direction of the negative gradient is known as the method of *steepest descent* or gradient descent. Steepest descent proposes a new point

$$\vec{x}' = \vec{x} - \varepsilon \nabla_{\vec{x}} f(\vec{x})$$

- ε is a positive scalar determining the size of the step. It is called *learning rate*
- Steepest descent converges when every element of the gradient is zero – at stationary points!

Jacobian

- To implement gradient descent from layer-to-layer of NN we need to compute a gradient of maps of the form $f: \mathbb{R}^n \to \mathbb{R}^m$, i.e. we need to compute partial derivatives of a function whose input and output are both vectors. The matrix containing all such partial derivatives is known as a ***Jacobian*** matrix, denoted $\mathbb{J} \in \mathbb{R}^{m\times \mathrm{n}}$ where $[\mathbb{J}]_{ij} = \frac{\partial}{\partial x_j}(\vec{f}(\vec{x}))_i$.

Example:

Let $\vec{f}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1^2 + x_1 x_2 \\ 3x_1 + x_2^2 x_1 \end{pmatrix}$ then

$\mathbb{J}(f) = ?$

Jacobian

- To implement gradient descent from layer-to-layer of NN we need to compute a gradient of maps of the form $f: \mathbb{R}^n \to \mathbb{R}^m$, i.e. we need to compute partial derivatives of a function whose input and output are both vectors. The matrix containing all such partial derivatives is known as a ***Jacobian*** matrix, denoted $\mathbb{J} \in \mathbb{R}^{m\times \mathrm{n}}$ where $[\mathbb{J}]_{ij} = \frac{\partial}{\partial x_j}(\vec{f}(\vec{x}))_i$.

Example:

Let $\vec{f}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1^2 + x_1 x_2 \\ 3x_1 + x_2^2 x_1 \end{pmatrix}$ then

$$\mathbb{J}(f) = \begin{pmatrix} \frac{\partial}{\partial x_1}(x_1^2 + x_1 x_2) & \frac{\partial}{\partial x_2}(x_1^2 + x_1 x_2) \\ \frac{\partial}{\partial x_1}(3x_1 + x_2^2 x_1) & \frac{\partial}{\partial x_2}(3x_1 + x_2^2 x_1) \end{pmatrix} =?$$

Jacobian

- To implement gradient descent from layer-to-layer of NN we need to compute a gradient of maps of the form $f: \mathbb{R}^n \to \mathbb{R}^m$, i.e. we need to compute partial derivatives of a function whose input and output are both vectors. The matrix containing all such partial derivatives is known as a ***Jacobian*** matrix, denoted $\mathbb{J} \in \mathbb{R}^{m \times \mathrm{n}}$ where $[\mathbb{J}]_{ij} = \frac{\partial}{\partial x_j}(f(\vec{x}))_i$.

Example:

Let $f\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1^2 + x_1 x_2 \\ 3x_1 + x_2^2 x_1 \end{pmatrix}$ then $\mathbb{J}(f) = \begin{pmatrix} 2x_1 + x_2 & x_1 \\ 3 + x_2^2 & 2x_2 x_1 \end{pmatrix}$

Chain Rule in Vector form

Gradient of a vector function:

Given $\vec{f}(\vec{x}) = \begin{pmatrix} f_1(\vec{x}) \\ \vdots \\ f_k(\vec{x}) \end{pmatrix}$ by definition $\nabla_{\vec{x}} f_i(\vec{x}) = \begin{pmatrix} \frac{\partial f_i}{\partial x_1} \\ \vdots \\ \frac{\partial f_i}{\partial x_n} \end{pmatrix}$ and $\nabla_{\vec{x}} \vec{f}(\vec{x}) =$ $(\nabla_{\vec{x}} f_1(\vec{x}) \quad \cdots \quad \nabla_{\vec{x}} f_k(\vec{x}))$ so $\nabla_{\vec{x}} \vec{f}(\vec{x}) = \mathbb{J}(\vec{f})^T$

Chain Rule: For a given $f(g_1(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n))$ holds

$$\frac{\partial f}{\partial x_i} = \sum_{j=1}^{k} \frac{\partial f}{\partial g_j} \cdot \frac{\partial g_j}{\partial x_i} = \nabla_{\vec{g}} f(\vec{g}) \cdot \begin{pmatrix} \frac{\partial g_1}{\partial x_i} \\ \vdots \\ \frac{\partial g_k}{\partial x_i} \end{pmatrix}$$

Then for a vector function $f\left(\vec{\mathrm{g}}(\vec{\mathrm{x}})\right)$ we obtain $\nabla_{\vec{x}} f = \mathbb{J}(\vec{g})^T \nabla_{\vec{g}} f$

Reading

- Ch. 1.3